1 Pendahuluan

2 Dekompisi secare rekursif

2.1 pendefinisian operasi dasar

3 Contoh DnC

3. Binary Search

3.1 karatsuba

Perkalian Matriks denga Devide and Conquer

Misalkan kita punya dua matriks A dan B dengan ukuran n x n. proses pembagian dilakukan dengan menjadi 4 sub matriks dengan ukuran n/2 x n/2. Untuk kemudahan kita andaikan $n = 2^k$ dengan k adalah bilangan bulat positif. Maka kita dapat menuliskan perkalian matriks A dan B sebagai berikut:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}.$$

Dengan begitu kita punya matriks C yang dapat dituliskan sebagai berikut:

$$\begin{aligned} \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} &= \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \\ &= \begin{pmatrix} A_{11} \cdot B_{11} + A_{12} \cdot B_{21} & A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \\ A_{21} \cdot B_{11} + A_{22} \cdot B_{21} & A_{21} \cdot B_{12} + A_{22} \cdot B_{22} \end{pmatrix} \end{aligned}$$

Dengan:

$$\begin{aligned} C_{11} &= A_{11} \cdot B_{11} + A_{12} \cdot B_{21}, \\ C_{12} &= A_{11} \cdot B_{12} + A_{12} \cdot B_{22}, \\ C_{21} &= A_{21} \cdot B_{11} + A_{22} \cdot B_{21}, \\ C_{22} &= A_{21} \cdot B_{12} + A_{22} \cdot B_{22} \end{aligned}$$

Perhatikan bahwa kita perlu melakukan 8 kali perkalian matriks $n/2 \times n/2$ untuk mendapatkan matriks C. Dengan menggunakan algoritma DnC. Dengan begitu persamaan rekursif runnig time dari algoritma perkalian matriks adalah sebagai berikut:

$$T(n) = 8T(n/2) + \Theta(1)) \quad n > 1, \quad T(1) = \Theta(1)$$

Perhatikan juga bahwa kompleksitas dari melakukan conquer adalah $\Theta(1)$ karena algoritma ini bersifat inplace. Dengan mensubtitusi $n = 2^k$ maka kita punya persamaan rekurensinya sebagai berikut:

$$t_k - 8t_{k-1} = 1$$

Dan mudah di tunjukkan bahwa perkalian matris memiliki order of growth $\Theta(n^3)$

```
def matmul_dnc(
    A: np.ndarray,
    B: np.ndarray,
    C: np.ndarray,
    n: int = None,
) -> np.ndarray:
    """
    matmul dnc on nxn matrix it's O(n^3)
    """
    if n == 1:
        C += A * B
        return
    n_init = n
    # if n not power of 2:
    if not isPowerOfTwo(n):
        n = max(A.shape[0], A.shape[1], B.shape[0], B.shape[1])
        n = 2 ** (n - 1).bit_length()
        A = np.pad(A, ((0, n - A.shape[0]), (0, n - A.shape[1])))

        B = np.pad(B, ((0, n - B.shape[0]), (0, n - B.shape[1])))
        C = np.pad(C, ((0, n - C.shape[0]), (0, n - C.shape[1])))

    # conquer
    A_11 = A[: n // 2, : n // 2]
    A_12 = A[: n // 2, n // 2 :]
    A_21 = A[n // 2 :, : n // 2]
    A_22 = A[n // 2 :, n // 2 :]
    B_11 = B[: n // 2, : n // 2]
    B_12 = B[: n // 2, n // 2 :]
    B_21 = B[n // 2 :, : n // 2]
    B_22 = B[n // 2 :, n // 2 :]
    C_11 = C[: n // 2, : n // 2]
    C_12 = C[: n // 2, n // 2 :]
    C_21 = C[n // 2 :, : n // 2]
    C_22 = C[n // 2 :, n // 2 :]
```

```
    matmul_dnc(A_11, B_11, C_11, n // 2)
    matmul_dnc(A_12, B_21, C_11, n // 2)
    matmul_dnc(A_11, B_12, C_12, n // 2)
    matmul_dnc(A_12, B_22, C_12, n // 2)
    matmul_dnc(A_21, B_11, C_21, n // 2)
    matmul_dnc(A_22, B_21, C_21, n // 2)
    matmul_dnc(A_21, B_12, C_22, n // 2)
    matmul_dnc(A_22, B_22, C_22, n // 2)

    C = np.vstack((np.hstack((C_11, C_12)), np.hstack((C_21, C_22))))

    if not isPowerOfTwo(n_init):
        C = C[:n_init, :n_init]
        return C

    return C
```

3.2 Strassen

Berdasarkan rangkuman materi brute-force, kita tahu bahwa perkalian matriks memiliki order of growth $\Theta(n^3)$. Namun pada tahun 1969, Volker Strassen menemukan algoritma yang dapat mengurangi order of growth menjadi $\Theta(2.807) = \Theta(n^{\log 7})$. Algoritma ini menggunakan 7 kali perkalian matriks $n/2 \times n/2$ untuk mendapatkan matriks C.

Strassen bisa mengurangi running time dari jumlah perkalian submatriks yang dibutuhkan menjadi 7, dengan sedikit trade-off pada saat melakukan conquer, yang menjadi $\Theta(n^2)$.

Jadi, persamaan rekursif dari algoritma Strassen adalah sebagai berikut:

$$T(n) = 7T(n/2) + \Theta(n^2)) \quad n > 1, \quad T(1) = \Theta(1)$$

Dengan master therorem karena $a = 7, b = 2, f(n) = \Theta(n^2)$maka kita punya kompleksitas waktunya $\Theta(n^{\log 7})$.

Ide dari penemuan algoritma strassen didasari pada fakta berikut:

$$x^2 - y^2 = x^2 - xy + xy - y^2 = x(x - y) + y(x - y) = (x + y)(x - y)$$

Perhatikan bahwa kita hanya membutuhkan 1 kali perkalian dan 2 kali operasi penjumlahan atau pengurangan untuk mendapatkan $x^2 - y^2$ pada form ruas kanan dibandingkan dengan 2 kali perkalian pada form ruas kiri.

Kita punya algoritma Devide and Conquer strassen sebagai berikut: